HistMan: A Class for Simplified Histogram Management

Brett Viren

Physics Department

BROOKHAVEN NATIONAL LABORATORY

MINOS Week In The Woods, 2005

Outline

BROOKHAVEN NATIONAL LABORATORY

BROOKHAVEN NATIONAL LABORATORY

What is HistMan

HistMan provides three primary features:

1. A simplified API to booking and filling histograms.
2. Organizes histograms in a hierarchy of TFolders.
3. Provides for simple I/O of part or all of this hierarchy.

In addition:

- Any TObject can be stored in this heirarchy
- The hierarchy can:
  - be held by the individual HistMan instance, or
  - attached to ROOT Memory and thus visible from a TBrowser.
- This hierarchy is addressed with the usual notion of a path.
- The HistMan written files can be directly browsed in a bare ROOT session w/out the HistMan library

BROOKHAVEN NATIONAL LABORATORY

Three Modes of Use

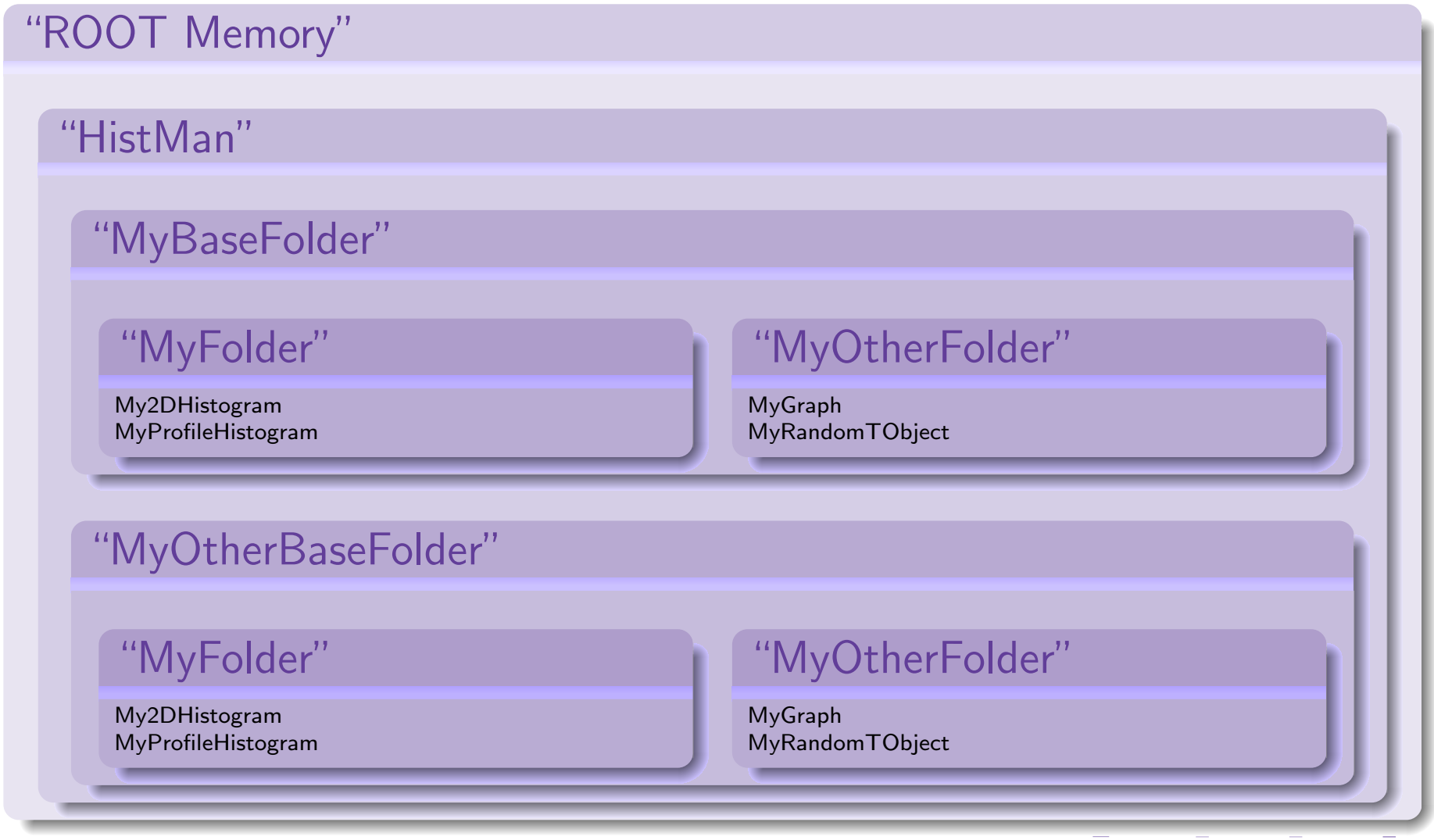
A HistMan instance has three modes of use, differentiated by how it was created:

1. Hierarchy is attached to "ROOT Memory"
2. Hierarchy is isolated to just this HistMan instance
3. Hierarchy is read from a file

Details next....

Mode 1: HistMan Attaches Heirarchy to ROOT Memory

In this mode HistMan adds its root folder, called "HistMan" to the "ROOT Memory" folder.



Mode 1: HistMan Attaches Heirarchy to ROOT Memory

Later HistMan instances can attach to just one part of this heirachy...

"ROOT Memory"

"HistMan"

"MyBaseFolder" `HistMan hm("MyBaseFolder")`

"MyFolder"

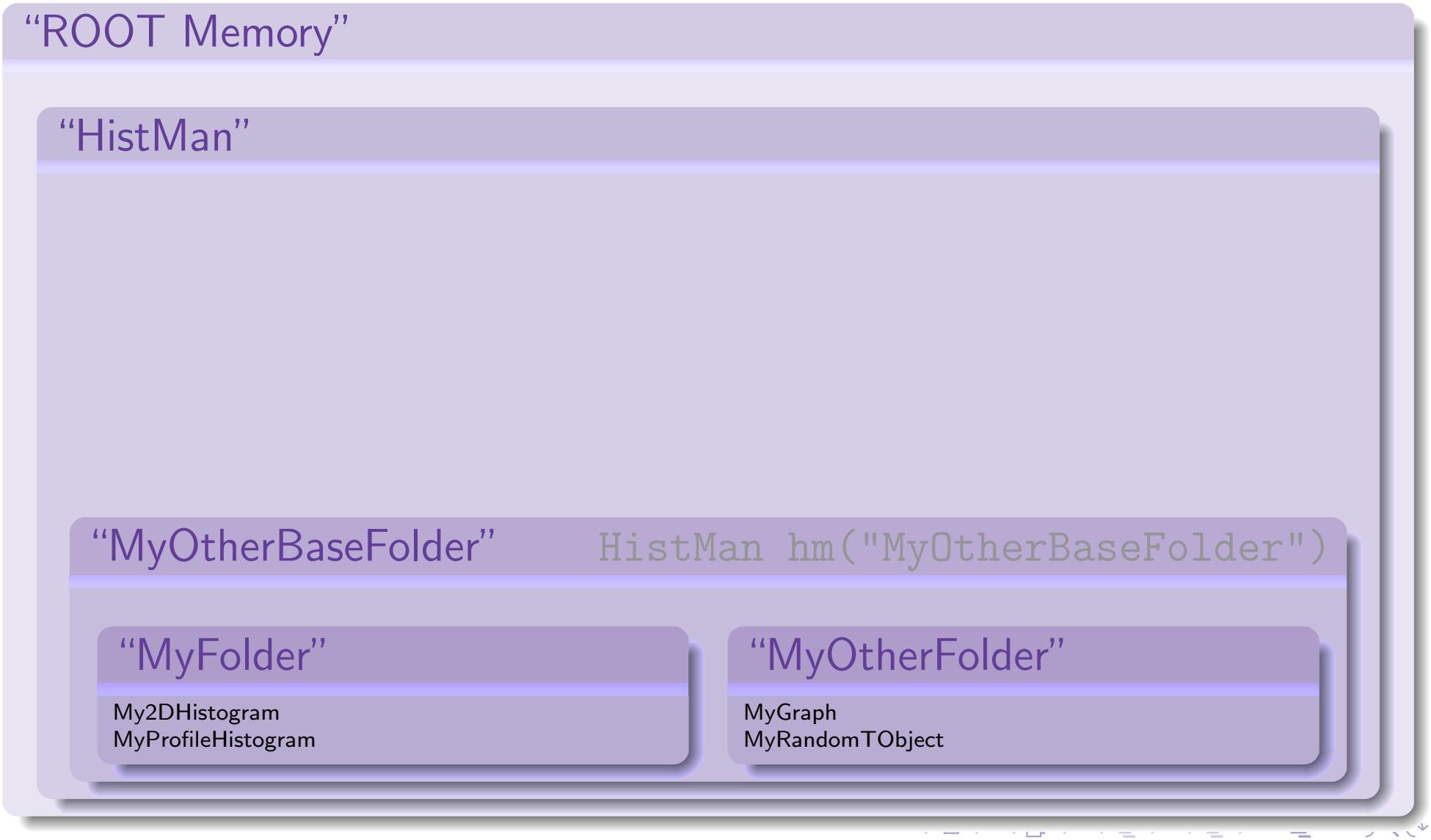
My2DHistogram
MyProfileHistogram

"MyOtherFolder"

MyGraph
MyRandomTObject

Mode 1: HistMan Attaches Heirarchy to ROOT Memory

...while the others remain hidden.



HistMan vs. the Hierarchy

It is important to understand how HistMan instances relate to the persistent hierarchy of TFolders and TObjects held in ROOT Memory.

1. HistMan instances are rooted in only one branch of the hierarchy, outside that branch is not accessible by the instance.
2. HistMan instances can come and go, the hierarchy remains.
3. This means multiple HistMan instance + the hierarchy actually implements a *Singleton* pattern.
4. There is very little need to pass around HistMan instances.
   - Pass around the path (or hard code it!) and create HistMan with that path when needed.
   - Caveat: obviously don't want to create HistMan instances inside a tight loop.

BROOKHAVEN NATIONAL LABORATORY

Mode 2: Heirarchy Left Unattached to ROOT Memory

In this mode each instance will hold a separate hierarchy. It can optionally be owned by the HistMan instance

"HistFolder0" (folder implicitly created in HistMan ctor)

"MyFolder"

My2DHistogram
MyProfileHistogram

"MyOtherFolder"

MyGraph
MyRandomTObject

"MyTopFolder" (folder explicitly passed to HistMan ctor)

"MyFolder"

My2DHistogram
MyProfileHistogram

"MyOtherFolder"

MyGraph
MyRandomTObject

At any time, this mode can become like Mode 1 by calling `HistMan::RegisterWithRoot`

BROOKHAVEN NATIONAL LABORATORY

Mode 3: HistMan given a TFile

In this mode the HistMan instance is created with a previously written. Here

1. The in-file TDirectory based hierarchy is converted into an in-memory TFolder based one and placed in the "HistMan" folder.
2. This "HistMan" folder is attached to "ROOT Memory", thus visible from a TBrowser in the exact same way as the original code that filled the original hierarchy.
3. The TFile must be kept open as the TObjects are still owned by the TFile.
4. More objects can be added to this hierarchy and even written out to a new file.

Note: once a HistMan instance is created in this mode it is essentially identical to Mode 1 as the Hierarchy is attached to "ROOT Memory".

BROOKHAVEN NATIONAL LABORATORY

BROOKHAVEN NATIONAL LABORATORY

Constructors, one for each usage mode

`HistMan(const char* path = "")` Create a `HistMan` instance rooted at the given **path**. Any thing added to the hierarchy through this instance will be visible at `//root/ROOT memory/HistMan/path/`.

`HistMan(TFolder* folder, bool own=true)` Create a `HistMan` instance with an explicit base folder. If "own" is true the folder is adopted. This will not attach the hierarchy to the "ROOT Memory" folder.

`HistMan(TFile& file)` Create a `HistMan` instance and pull out a hierarchy from a `TDirectory` called "*HistMan*" from the file (such as would exist if the file was created by **HistMan::WriteOut()**). If fail, will act like **Histman("")** above. If successfull, the file continues to own objects in the hierarchy so must be kept open while they are to be used.

BROOKHAVEN NATIONAL LABORATORY

Using a HistMan to Book Histograms

```
// 1D-like histograms
template<class THType>
THType* Book(const char* name, const char* title,
             int nbinsx, Axis_t xmin, Axis_t xmax,
             const char* path=".", Bool_t sumw2=kFALSE);

// 2D-like histograms
template<class THType>
THType* Book(const char* name, const char* title,
             int nbinsx, Axis_t xmin, Axis_t xmax,
             int nbinsy, Axis_t ymin, Axis_t ymax,
             const char* path=".", Bool_t sumw2=kFALSE);
// Examples:
HistMan hm(``myfolder'');
hm.Book<TH1D>(``ereco'',''Reco Neutrino Energy'',100,0,10);
hm.Book<TH2D>(``hadmon'',''Hadron Monitor Display'',
              7,-40,40,7,-40,40);
```

BROOKHAVEN NATIONAL LABORATORY

Using a HistMan to Fill Histograms

`bool Fill1d(const char* pathname, Axis_t x, Stat_t w=1.0)` Lookup a 1D histogram by pathname="path/name" and Fill() it. If lookup or any casting fails, false is returned and an error message is printed at Msg level Warning.

`bool Fill2d(const char* pathname, Axis_t x, Axis_t w, Stat_t w=1.0)` Lookup a 2D histogram by pathname="path/name" and Fill() it. If lookup or any casting fails, false is returned and an error message is printed at Msg level Warning.

BROOKHAVEN NATIONAL LABORATORY

Explicit Access of the Hierarchy's Contents

You can place and retrieve arbitrary TObject derived instances into the HistMan hierarchy.

1. Place an object in the hierarchy:

   ```
   TObject* HistMan::Adopt(const char* path, TObject* o);
   ```

   Will return NULL (and the object will be deleted) if an object of the same name already exists, o.w. the object is returned. Passing in an empty string places the object in this HistMan instance's top level folder.

2. Retrieve an object in a type-safe manner:

   ```
   template<class THType>
   THType* HistMan::Get(const char* path);
   ```

   Return the object with the given type and at given path in this instance's branch of the hierarchy, o.w. 0 is returned.

BROOKHAVEN NATIONAL LABORATORY

BROOKHAVEN NATIONAL LABORATORY

Working around ROOT's flat histogram lookup

- By default, ROOT maintains a behind-the-scenes lookup of histograms based on their names.
- This is a flat namespace, which makes it possible to have name clashes.
- Name clashes are especially likely when multiple instances of the same module exist in a job.

To avoid this, in your JobCModule, follow this example:

```
void MyModule::BeginJob() {
    HistMan hm(this->GetUniqueName());
    hm.Book<TH1D>(...);
}
JobCResult MyModule::Ana(const MomNavigator *mom) {
    HistMan hm(this->GetUniqueName());
    hm.Fill1d(...);
}
```

BROOKHAVEN NATIONAL LABORATORY

Requirements for BeamData "online" Monitor Process

Spot the unifying theme:

- Avoid the monolithic design of the DAQ `Monitoring` package. (I'm lazier than David, I don't want to have to manage other's contributions!)
- Avoid writing any plot consumer code by leverage existing `CDFMonitoringFwk` GUI and plot server code.
- But, not **need** the CDFMonitoringFwk to make development and testing of the plot generators easier.
- Avoid having to understanding the details of the dispatcher client API.
- Use something familiar to Minos programers to lower the bar to bringing **other** people into generating the plots. (Did I mention I'm lazy?)

BROOKHAVEN NATIONAL LABORATORY

The Design of BeamDataMonitoring

Solution:

- Plot generators stuff fully built graphical TObjects (typically filled TCanvas) into a HistMan, don't care what happens to them.
- Use **loon** as opposed to some custom main program. This gives both file and dispatcher I/O for free.
- Generate the plots in multiple standard JobCModules. Simple way for multiple contributions w/out stepping on toes.
- Had to do **some** work: wrote a special JobCModule, `CDFMonitoringModule` which is placed at the end of the job path. It digs into the HistMan hierarchy and pushes anything found in a special "Monitoring" folder on to the `CDFMonitoringFwk`'s `Server`.
- Alternatively, write to a file by placing `HistMan::WriteOut` after `JobC::Run` or `DataUtil/HistManModule` at the end of the path. Then click around TBrowser in a bare root session to test the plots.

BROOKHAVEN NATIONAL LABORATORY

Summary

HistMan

- provides a convenient and flexible way to organize and access histograms and other TObjects.
- presents a simplified histogram API.
- provides a symmetric I/O mechanism.
- allows us to collect all the histograms from disparate (and potentially duplicate) job modules without name clashes.
- useful for simplifying day-to-day analysis jobs and for production infrastructure.

BROOKHAVEN NATIONAL LABORATORY